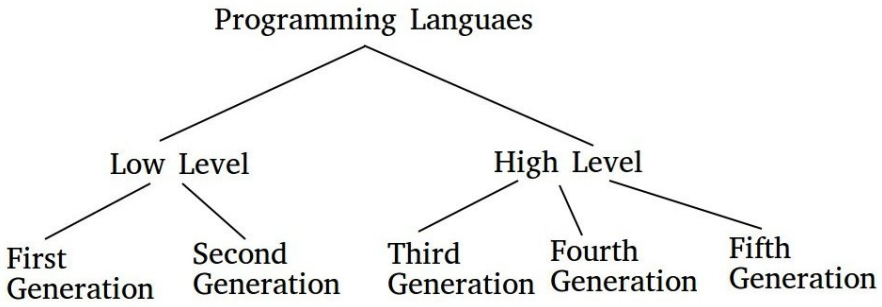
UNIT-1

GENERATIONS OF PROGRAMMING LANGUAGES

The first two generations are called low-level languages. The next three generations are called high-level languages.



1. First-Generation Language :

The first-generation languages are also called machine languages/ 1G language. This language is machine-dependent. The machine language statements are written in binary code (0/1 form) because the computer can understand only binary language.

Advantages :

1. Fast & efficient as statements are directly written in binary language.
2. No translator is required.

Disadvantages :

1. Difficult to learn binary codes.
2. Difficult to understand – both programs & where the error occurred.

2. Second Generation Language :

The second-generation languages are also called assembler languages/ 2G languages. Assembly language contains human-readable notations that can be further converted to machine language using an assembler.

Assembler – converts assembly level instructions to machine-level instructions.

Programmers can write the code using symbolic instruction codes that are meaningful abbreviations of mnemonics. It is also known as low-level language.

Advantages :

1. It is easier to understand if compared to machine language.
2. Modifications are easy.
3. Correction & location of errors are easy.

Disadvantages :

1. Assembler is required.
2. This language is architecture /machine-dependent, with a different instruction set for different machines.

3. Third-Generation Language :

The third generation is also called procedural language /3 GL. It consists of the use of a series of English-like words that humans can understand easily, to write instructions. It's also called High-Level Programming Language. For execution, a program in this language needs to be translated into machine language using a Compiler/ Interpreter. Examples of this type of language are C, PASCAL, FORTRAN, COBOL, etc.

Advantages :

1. Use of English-like words makes it a human-understandable language.
2. Lesser number of lines of code as compared to the above 2 languages.
3. Same code can be copied to another machine & executed on that machine by using compiler-specific to that machine.

Disadvantages :

1. Compiler/ interpreter is needed.

2. Different compilers are needed for different machines.

4. Fourth Generation Language :

The fourth-generation language is also called a non – procedural language/ 4GL. It enables users to access the database. Examples: SQL, Foxpro, Focus, etc.
These languages are also human-friendly to understand.

Advantages :

1. Easy to understand & learn.
2. Less time is required for application creation.
3. It is less prone to errors.

Disadvantages :

1. Memory consumption is high.
2. Has poor control over Hardware.
3. Less flexible.

5. Fifth Generation Language :

The fifth-generation languages are also called 5GL. It is based on the concept of artificial intelligence. It uses the concept that rather than solving a problem algorithmically, an application can be built to solve it based on some constraints, i.e., we make computers learn to solve any problem. Parallel Processing & superconductors are used for this type of language to make real artificial intelligence.
Examples: PROLOG, LISP, etc.

Advantages :

1. Machines can make decisions.
2. Programmer effort reduces to solve a problem.
3. Easier than 3GL or 4GL to learn and use.

Disadvantages :

1. Complex and long code.

UNIT-5

FILES

Introduction to Files in C:

A File is a collection of data stored in the secondary memory. So far data was entered into the programs through the keyboard. So Files are used for storing information that can be processed by the programs. Files are not only used for storing the data, programs are also stored in files. In order to use files, we have to

learn file input and output operations. That is, how data is read and how to write into a file.

Types of Files in a C Program

When referring to file handling, we refer to files in the form of data files. Now, these data files are available in 2 distinct forms in the C language, namely:

- Text Files
- Binary Files

Text Files

The text files are the most basic/simplest types of files that a user can create in a C program. We create the text files using an extension *.txt* with the help of a simple text editor. In general, we can use notepads for the creation of .txt files. These files store info internally in ASCII character format, but when we open these files, the content/text opens in a human-readable form.

Binary Files

The binary files store info and data in the binary format of 0's and 1's (the binary number system). Thus, the files occupy comparatively lesser space in the storage. In simpler words, the binary files store data and info the same way a computer holds the info in its memory. Thus, it can be accessed very easily as compared to a text file.

Why Do We Need File Handling in C?

- **Reusability:** File handling allows us to preserve the information/data generated after we run the program.
- **Saves Time:** Some programs might require a large amount of input from their users. In such cases, file handling allows you to easily access a part of a code using individual commands.
- **Commendable storage capacity:** When storing data in files, you can leave behind the worry of storing all the info in bulk in any program.
- **Portability:** The contents available in any file can be transferred to another one without any data loss in the computer system. This saves a lot of effort and minimises the risk of flawed coding.

Operations on the file:

The process of file handling enables a user to update, create, open, read, write, and ultimately delete the file/content in the file that exists on the C program's local file system. Here are the primary operations that you can perform on a file in a C program

- Naming the file.

- Opening the file.
- Reading from the file.
- Writing into the file.
- Closing the file.

1. Using Files in C:

To use a file four essential actions should be carried out. These are,

a. Declare a file pointer variable.
b. Open a file using the fopen() function.
c. Process the file using suitable functions.
d. Reading and Writing files
e. Close the file using the fclose() and fflush() functions.

Declaration of file pointer

A pointer variable is used to points a structure FILE. The members of the FILE structure are used by the program in various file access operation, but programmers do not need to concerned about them.

Syntax:-

```
FILE
*file_pointer_name;
```

Example :

```
FILE *ptr;
```

Opening a file

To open a file using the fopen() function. It is used to file open in the different modes.

Syntax:-

```
File_pointer_name=fopen("PATH"," MODE");
```

In the above syntax:

a. File_pointer_name is the pointer name.
b. PATH is the source link(drive,directory,filename).
c. MODE represents the different scenario's such as write,read and append etc…

Ex:-

```
ptr=fopen ("c:\\data\\example.txt"," w+");
```

Mode	Description
r	Opens a text file in reading mode
w	Opens or create a text file in writing mode
a	Opens a text file in append mode
r+	Opens a text file in both reading and writing mode

w+	Opens a text file in both reading and writing mode
a+	Opens a binary file in reading mode
rb	Opens a binary file in reading mode
wb	Opens or create a binary file in writing mode
Ab	Opens a binary file in append mode
rb+	Opens a binary file in both reading and writing mode
wb+	Opens a binary file in both reading and writing mode
ab+	Opens a binary file in both reading and writing mode

Writing into file

writing the data into the file then uses fprintf() function and “w” mode

Eg:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
Main()
{
        char name[25];
        FILE *ptr;
        Clrscr();
        Ptr=fopen(“c:\\venkat.txt”,”w”);
        If(ptr==NULL)
        {
                Printf(“FILE CREATION ERROR!”);
        }
        Printf(“ENTER THE DATA”);
        Scanf(“%s”,name);
        fpritf(ptr,“%s”,name);
        fclose(ptr);
        getch();
}
```

Reading from file

Reading the data from the file then uses fscanf() function and “r” mode

Eg:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
Main()
{
        char name[25];
        FILE *ptr;
        Clrscr();
        Ptr=fopen(“c:\\venkat.txt”,”r”);
        If(ptr==NULL)
        {
                Printf(“FILE CREATION ERROR!”);
        }
        Printf(“ENTER THE DATA”);
        Scanf(“%s”,name);
        fscanf(ptr,“%s”,name);
        fclose(ptr);
        Getch();
}
```

Closing Files

The file must be closed using the fclose() function. It is used to close the file.

Syntax:

Fclose(File_pointer_name);

Ex:

Fclose(ptr);

The argument fp is the FILE pinter associated with the stream. Fclose() returns 0 on success and -1 on error.